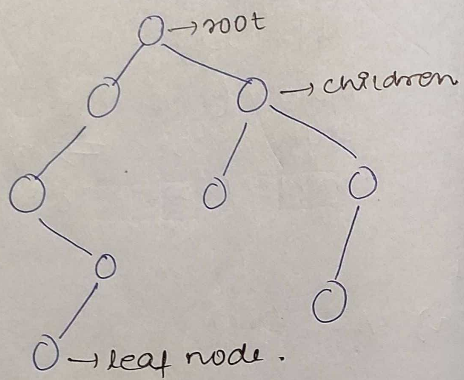
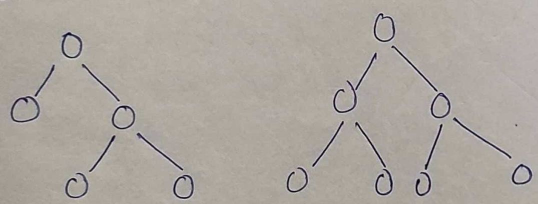
Ancestor



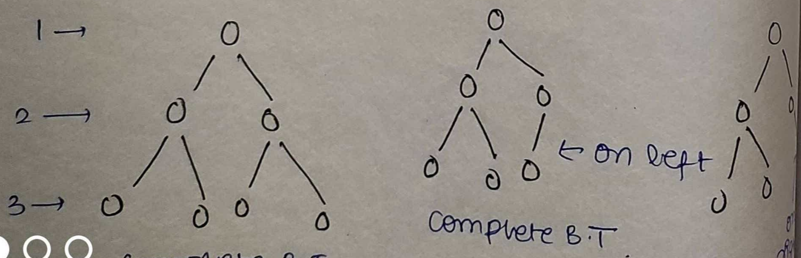
Types of Binary Tree :-

① Full B.T → either has 0 or 2 children.



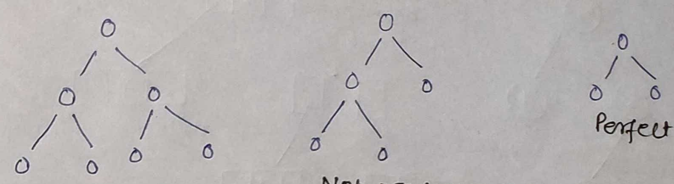
② Complete B.T → (i) all levels are completely filled except the last level

(ii) the last level has all nodes in left as possible.



Complete B.T   Complete B.T   not complete

③ Perfect B.T → all leaf nodes are at the same level.

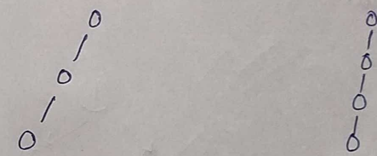


Perfect B.T   Not perfect B.T.   Perfect

④ Balanced B.T. → height of tree at max log(N)

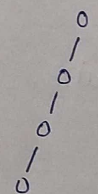
N → Nodes

$n = 8$, $\log_2 8 = 3$.

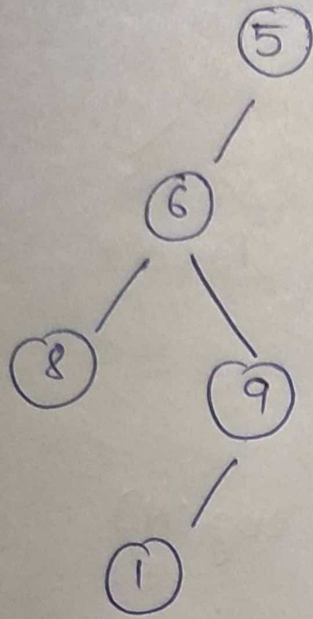


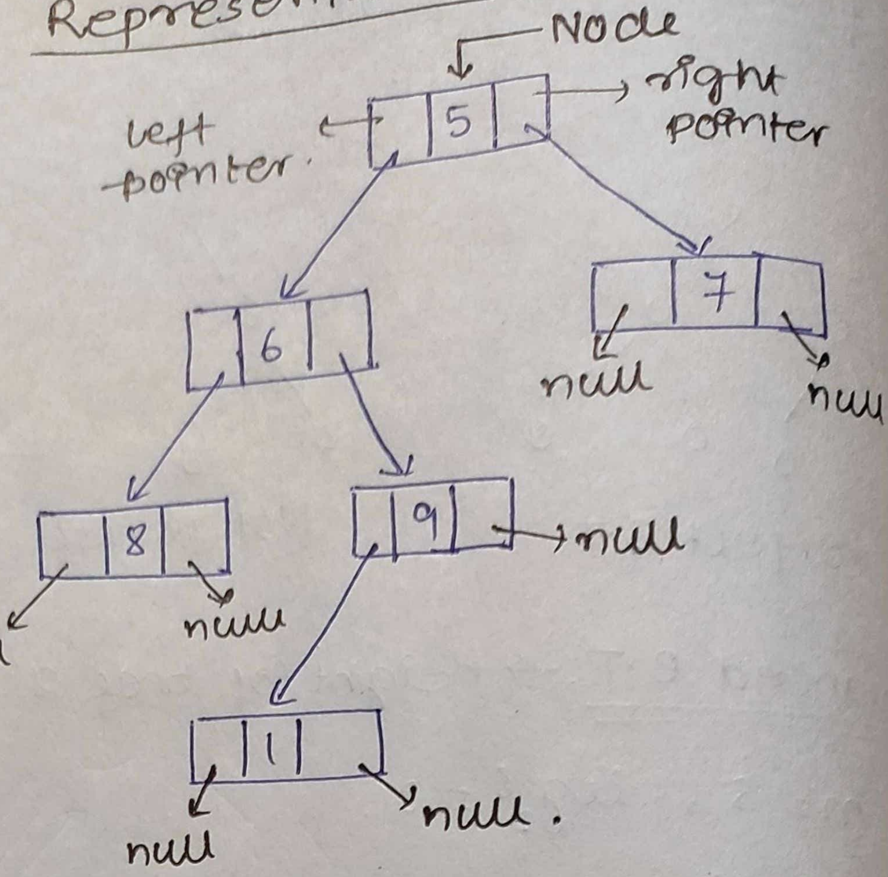
Balanced B.T   Not Balanced B.T

⑤ Degenerate Tree :- $n = 4$ Every Node has only single children.

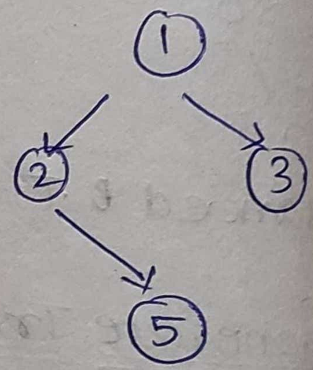


Binary Tree Representation

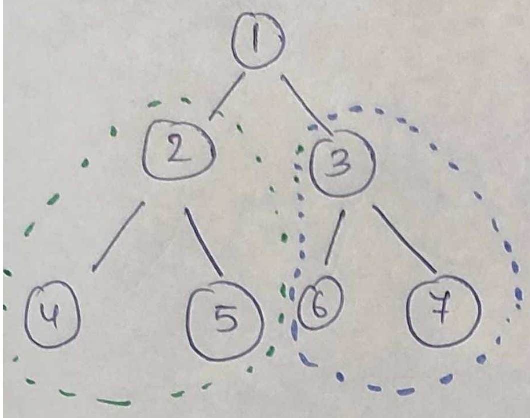





```
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
    Node (int val) {
        data = val;
        left = right = null;
    }
main() {
    struct Node * root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->right = new Node(5);
}
```



Traversal Techniques (BFS/DFS)



→ Inorder traversal (Left Root Right)

4 2 5 1 6 3 7

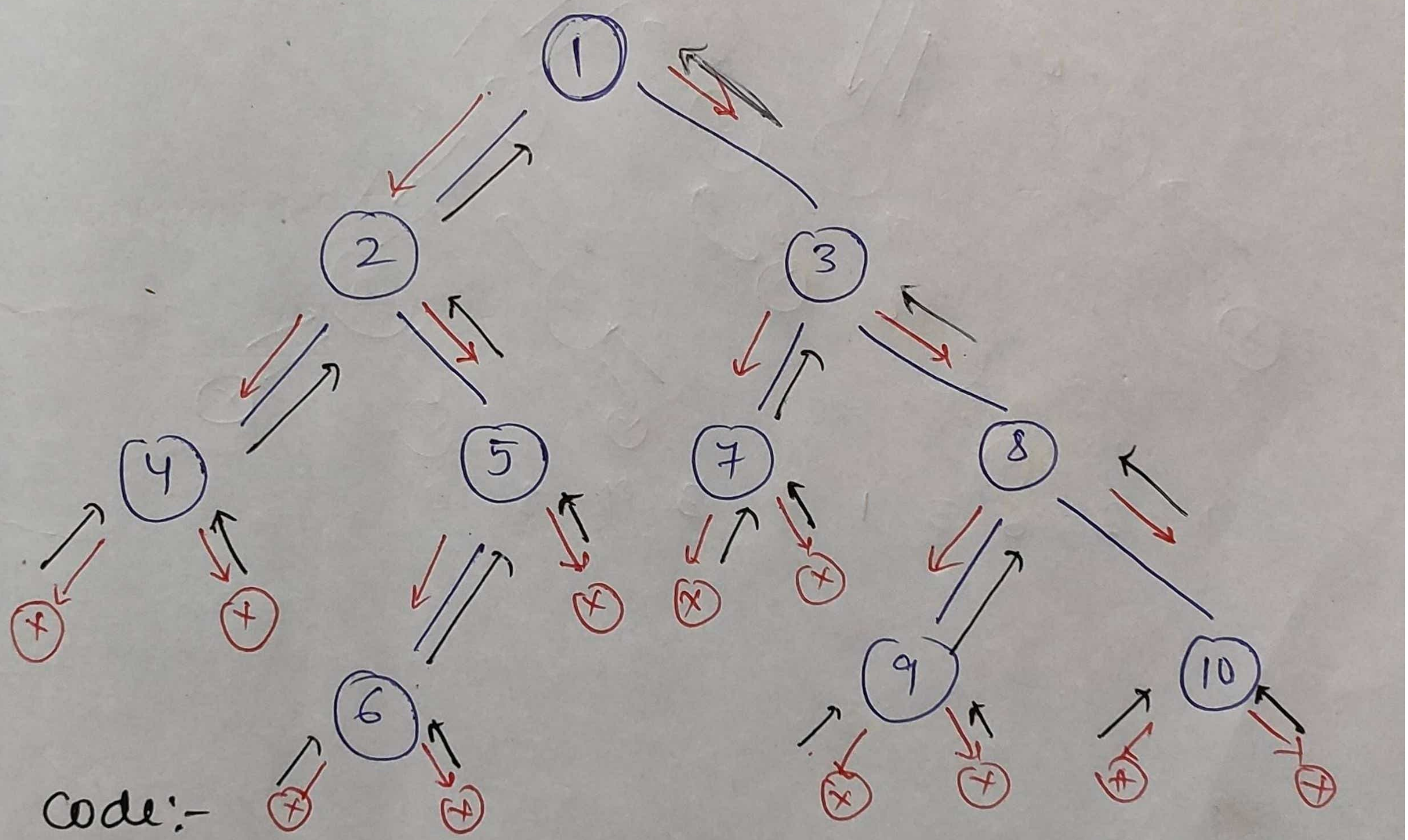
→ Pre-Order traversal (Root Left Right)

1 2 4 5 3 6 7

→ Post-Order Traversal (Left Right Root).

4 5 2 6 7 3 1

Implementation of Pre-Order Traversal (Root Left Right)



1 2 4 5 6 3 7 8 9

Code:-

```
void PreOrder(node) {
if (node == NULL) return;
print (node -> data);
preorder (node -> left);
preorder (node -> right);
```

$\boxed{\begin{array}{l} TC: O(n) \\ SC: O(n) \end{array}}$

Implementation of Inorder Traversal.

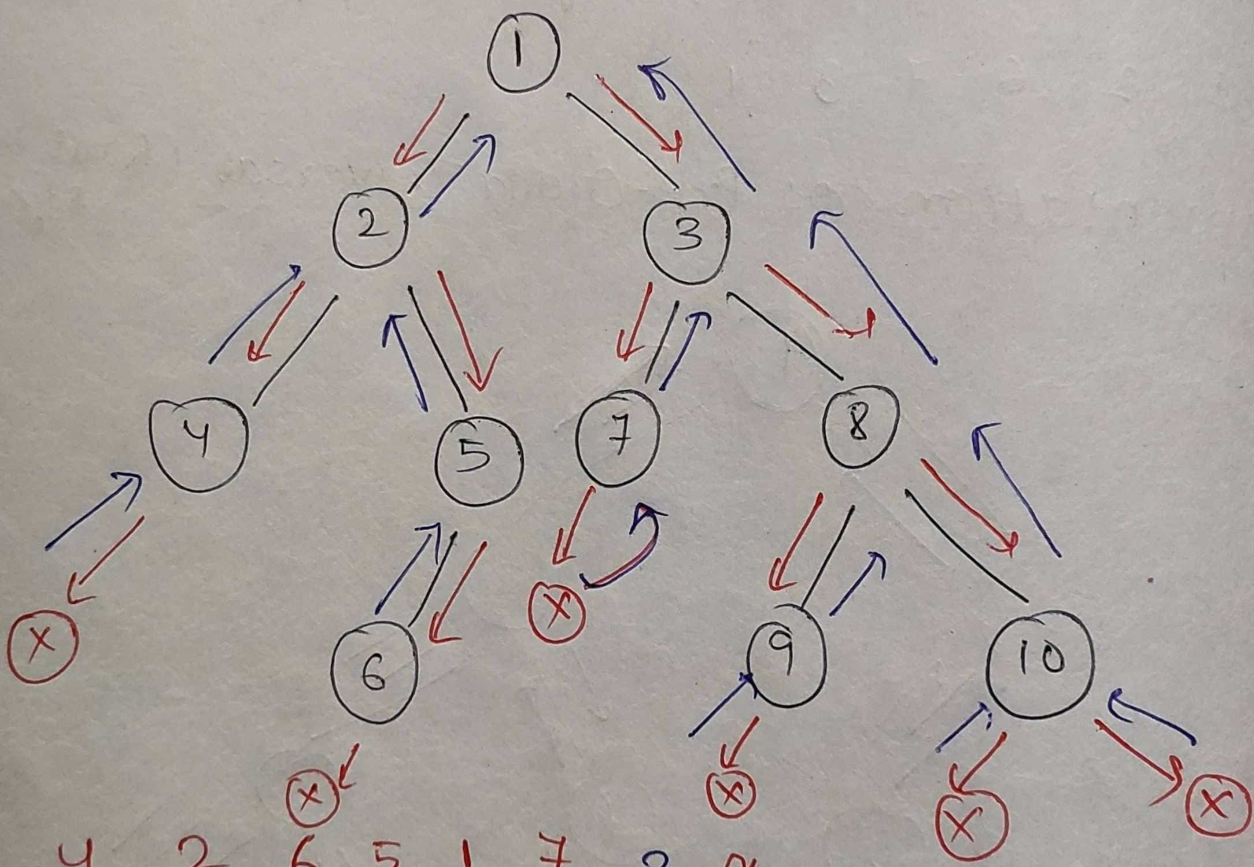
root

```
void inorder(node){

    if(node==null) return;

    inorder(node->left);
    print(node->data);
    inorder(node->right);

}
```

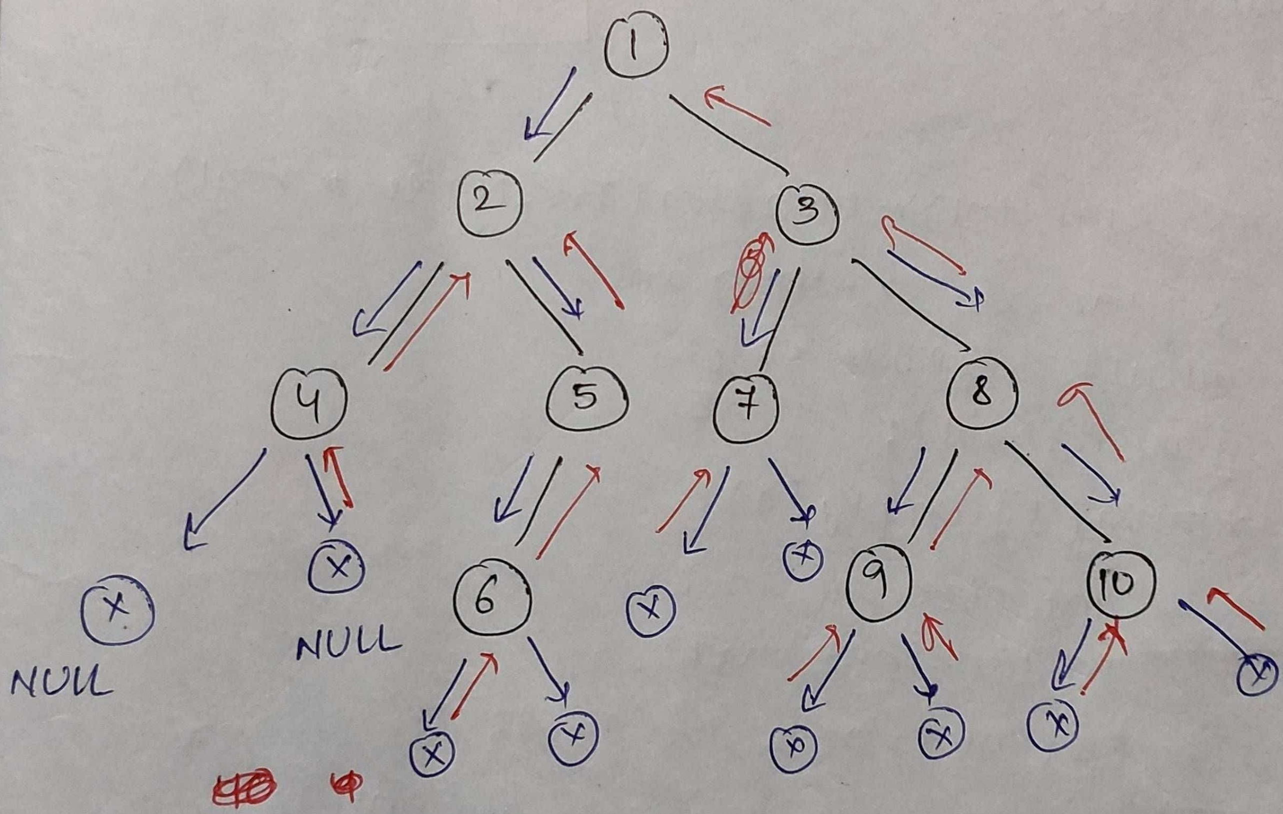


4 2 6 5 1 7 3 9 8 10

Implementation of Post-Order Traversal.

(Left ~~Root~~ Right) Root .

```
void PostOrder(node){
    if(node == NULL) return;
    PostOrder (node -> left);
    PostOrder (node -> right);
    print (node -> data) ;
}
```



4 6 5 2 . 7 9 10 8 3 1